Solving Sudoku With An Improved Strategy Based On Sparse Optimization Method

May 17, 2019

Luobin Wang, Junhao Zhang, Shuyi Ye, Hongyang Cai

Setup

With a given sudoku problem, we first encode it into a 1x729 binary vector, in which each 9-element piece relatively represents a cell in the original problem. The cells with clues are encoded based on Table 1, and the cell waiting to fill is represented by nine 0s.

Table 1. The integer numbers 1 to 9 are coded by a nine-dimensional binary vector.

Integer numbers	Binary vector	Integer numbers	Binary vector	Integer numbers	Binary vector
1	(1,0,0,0,0,0,0,0,0)	4	(0,0,0,1,0,0,0,0,0)	7	(0,0,0,0,0,0,1,0,0)
2	(0,1,0,0,0,0,0,0,0)	5	(0,0,0,0,1,0,0,0,0)	8	(0,0,0,0,0,0,0,1,0)
3	(0,0,1,0,0,0,0,0,0)	6	(0,0,0,0,0,1,0,0,0)	9	(0,0,0,0,0,0,0,0,1)

Then we translate the constraints of the given sudoku puzzle based on the binary vector we get from the first step. In every sudoku puzzle, it requires no repeated number in each row, column or box, and each grid should be filled. We transfer those constraints into linear constraints. We set x, a 729 dimension binary vector, to be our solution.

Let x_{ijk} represent the event that the (i, j) element of the Sudoku grid contains k, if it is true, $x_{ijk} = 1$, otherwise $x_{ijk} = 0$. Then the constraints are

- Column, $\sum_{i=1}^{9} x_{ijk} = 1$ for $1 \leq j, k \leq 9$
- Row, $\sum_{j=1}^{9} x_{ijk} = 1$ for $1 \leq i, k \leq 9$
- Box, $\sum_{j=3p-2}^{3p} \sum_{i=3q-2}^{3q} x_{ijk} = 1$ for $1 \leq k \leq 9$ and $1 \leq p, q \leq 3$.
- Grid, $\sum_{k=1}^{9} x_{ijk} = 1$ for $1 \leq i, j \leq 9$.
- Clues, should be given from the problem.

For example, the first column should contain all of the numbers from 1 to 9 without repeating. In x, we should have a "1" at first 9 entries of every 81 entries. We construct as I_{9*9} denotes the 9*9 identity matrix, and 0_{9*72} denotes a 9*72 matrix contains all zero.

$\underbrace{(I_{9\times9}0_{9\times72}I_{9\times9}0_{9\times72}\cdots I_{9\times9}0_{9\times72})}_{9}x = 1_{9\times1}.$ Similarly, we constructed constraints like this for each row, 3 by 3 box, cell, and clue as follows.

1st Roll: $\underbrace{(I_{9\times9}I_{9\times9}\cdots I_{9\times9}}_{9}0_{9\times648})x = 1_{9\times1}.$

1st Box: $(I_{9\times9}I_{9\times9}I_{9\times9}0_{9\times54}I_{9\times9}I_{9\times9}I_{9\times9}0_{9\times54}I_{9\times9}I_{9\times9}I_{9\times9}0_{9\times540})x = 1_{9\times1}$

1st Cell: $(\underbrace{11\cdots1}_{9}\underbrace{00\cdots0}_{720})x = 1.$

A clue: $(\underbrace{00\cdots0}_{144}\ 010000000\ \underbrace{00\cdots0}_{576})x = 1$

And in all, combining each constraint will construct a (324 + N) x 729 (N denotes the number of original clues) linear equality constraints.

Weighted LP1

We set $x_{ori} = x_1 - x_2$, and construct a completely new xs combining x_1 and x_2 (which will be the solution for WP1) in order to control that all entries in x are greater than 0. At the same time, we add weight to each element by multiplying a matrix $W=diag(w_1, w_2, \ldots, w_n)$ with

$$w_k = \frac{1}{|x_k|^{i-1}+\epsilon}, \quad 0 < \epsilon < 1$$

Since our xs is in 729*2=1458 dimensions, we reconstruct W=[W,W]. After several times of attempts, we decide to set epsilon equal to 0.5, since it gives the highest success rate.

(Weighted LP1) Solving the weighted ℓ_1-norm minimization problem (WP_1) by (LP1)

Input: $L = 10, \hat{x} = 0, \check{x} = 0, x_{ori} = \hat{x} - \check{x}, tol = 1 \times 10^{-10}$;

For i = 1:L

$W = \mathrm{diag}\left(\frac{1}{|x_{ori}|+\epsilon}\right)$;

Based on the method (LP1), $x_{new} = \hat{x} - \check{x}$ is obtained by solving the following linear programming problem:

$$\min W\begin{bmatrix}\hat{x}\\ \check{x}\end{bmatrix}, \text{s.t. } [A \quad -A]\begin{bmatrix}\hat{x}\\ \check{x}\end{bmatrix} = b, \ \begin{bmatrix}\hat{x}\\ \check{x}\end{bmatrix} \in C;$$

if $\|x_{new} - x_{ori}\| < tol$

break;

else

$x_{ori} = x_{new}$;

End

End

Output: x_{new}

Improved Strategy

After solving for the initial solution, we check if the solution satisfies the constraint by iterating over rows, columns, and boxes to see if there are repeats. If there are repeats, the solver will replace all the repeats with 0 and solve it again. If the solution we get still have repeated numbers, we repeat this step again.

If the above two steps fail again, then we use the "Successive Increase Solver"(SIS) designed in the paper to solve again. This solver will return back to the original problem and randomly add one more clue, which is chosen from the solution we get and is not repeated in the solution.

After solving the advanced problem coming from SIS, if the solution has repeated numbers again, we designed one more solver to handle it. We call it "the fourth solver". In the solver, we compare the solution from SIS and original solution, replace the cells that have the

same number in both solutions with 0. Then we randomly pick a cell from the original solution that is not 0 and add it to the original problem as a clue. After solving for it, if the solution fails the sanity check, we declare the failure.

The basic idea of the fourth solver is to separate the cells of the original solution into two disjoint sets of cells so that the cells in each set have a stronger relation with each other than those in the other set ("stronger relation between A and B" means the number filled in cell A by the program have stronger effect on the number that will be filled in cell B. Then SIS and the fourth solver will represent solutions based on different sets of cells. From the result, we observe the fourth solver indeed handled some failures from ISI.

Results

Small 1:

```
Aver Time:    2.69 secs. Success rate: 24 / 24
```

Small2:

```
Aver Time:    6.29 secs. Success rate: 753 / 1000
```

Large1:

```
Aver Time:    1.32 secs. Success rate: 957 / 1000
```

Large2:

```
Aver Time:    1.91 secs. Success rate: 1000 / 1000
```

Future Work

The strategy we use in this program does not have a satisfiable performance when dealing with the sudoku problems with few clues. However, based on the improvement from ISI to the fourth solver, we believe that the performance can be better if we find a more logistic and reliable way to add clue to the original problem instead of randomly pick a clue from the previous solution.

Reference

Tang, Y., Wu, Z., & Zhu, C. (n.d.). A Warm Restart Strategy for Solving Sudoku by Sparse Optimization Methods. *ArVix*. Retrieved from https://arxiv.org/pdf/1507.05995.pdf.